JavaScript 4: Frameworks

Chapter 20

Randy Connolly and Ricardo Hoar

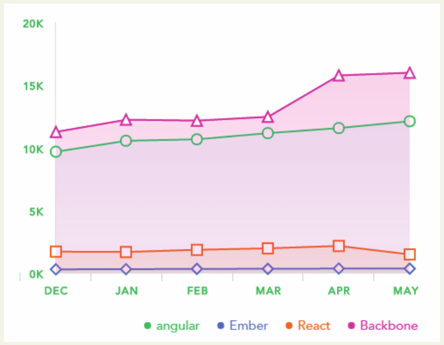
Fundamentals of Web Development

Chapter 20

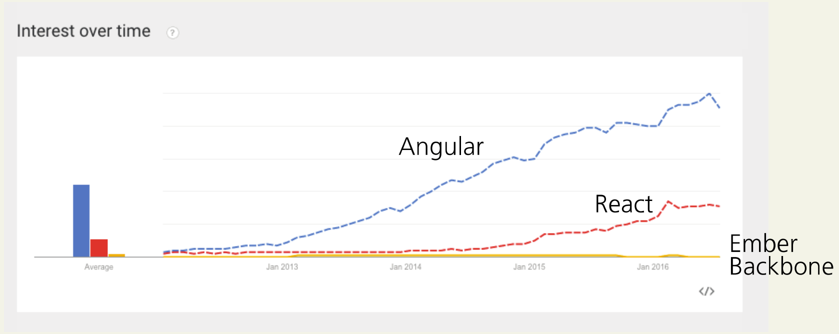
Chapter 20

JavaScript Frameworks

Popularity of Frameworks



from libscore.com



from Google Trends

	Angular	Backbone	Ember	React
Stackoverflow questions	187K	20K	19K	19K
Stackshare stacks	3.7K	1.3K	0.4K	1.7K
Github stars	51K	25K	16K	46K

JavaScript Frameworks

JavaScript Front-End Frameworks

- **Ember** forces developers to adopt a known and well-regarded approach to structuring and implementing a web application. It uses a variant of the MVC pattern
- **Angular** has many similarities to Ember (i.e., models, templates, and routing), and has the added advantage of being partially maintained by Google.
- **React** is a newer framework developed by Facebook. Unlike Ember and Angular, React is not a complete MVC-like framework; instead it focuses on the view.

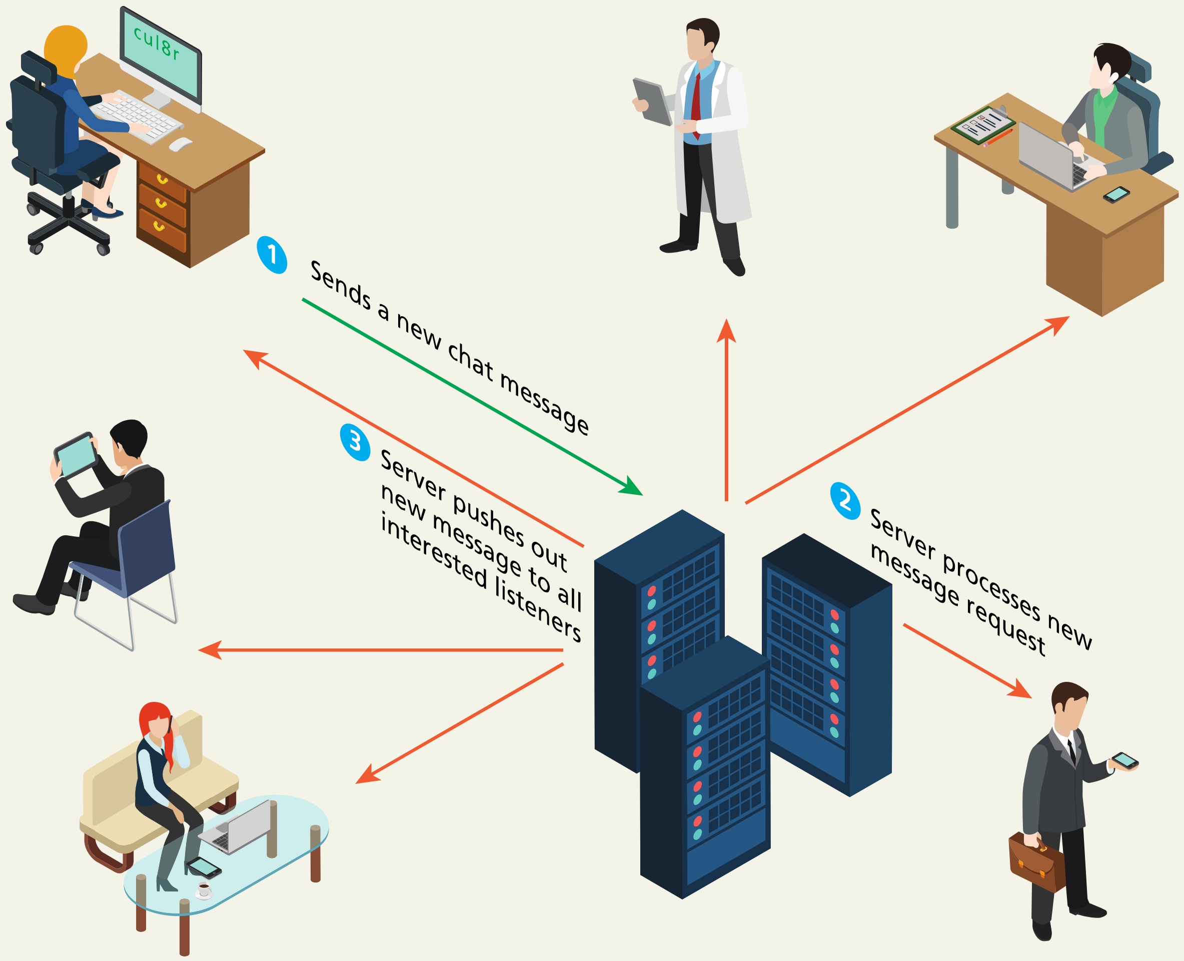
JavaScript Frameworks

- Node.js
- Alternative to LAMP stack
- MEAN stack
  - MongoDB-Express-Angular-Node.js

Chapter 20

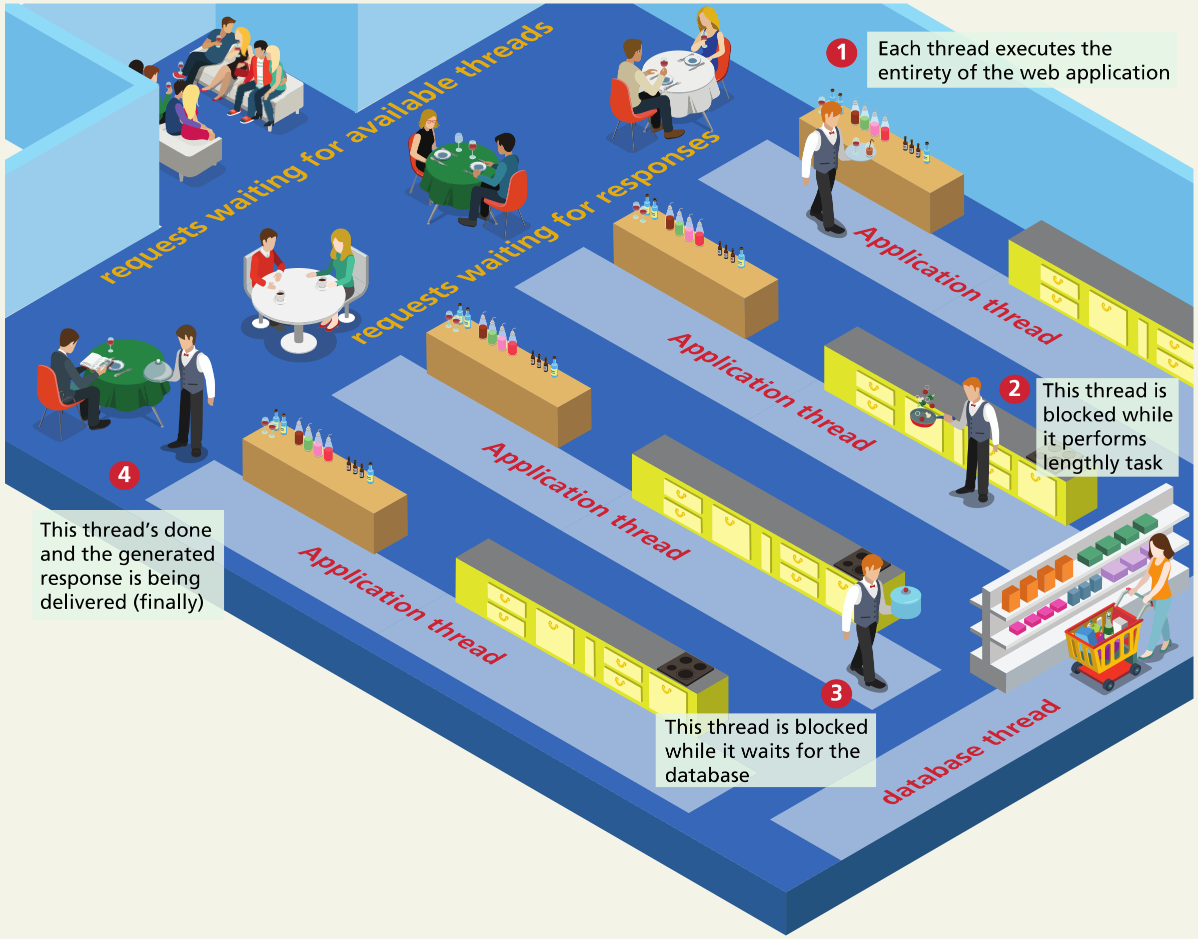
Node.js

Push based web application


cul8r
1 Sends a new chat message
2 Server processes new message request
3 Server pushes out new message to all interested listeners

Node.js

Blocking thread-based architecture (how apache /PHP run



1 Each thread executes the entirety of the web application

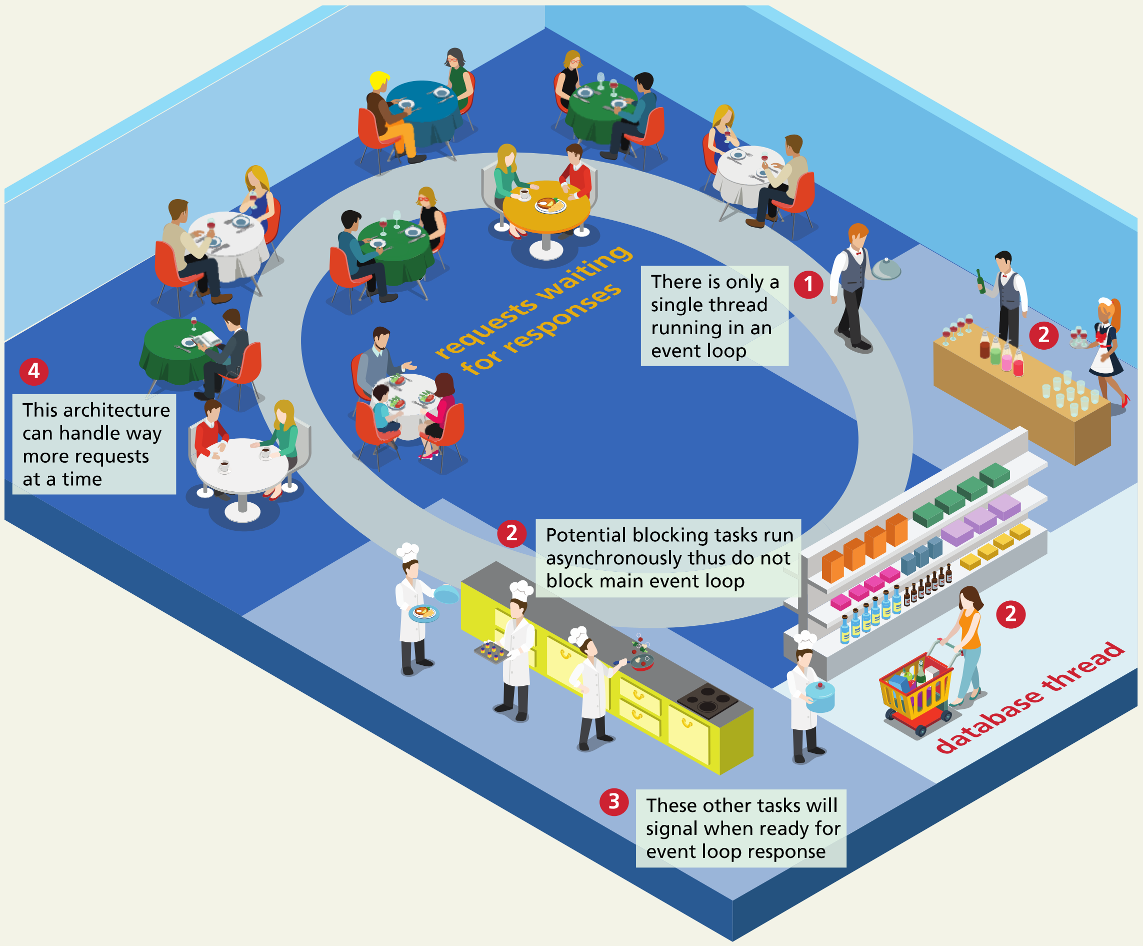
2 This thread is blocked while it performs lengthly task

3 This thread is blocked while it waits for the database

4 This thread's done and the generated response is being delivered (finally)

Node.js

Node.js single-thread architecture


requests waiting for responses
1 There is only a single thread running in an event loop
2
4 This architecture can handle way more requests at a time
2 Potential blocking tasks run asynchronously thus do not block main event loop
2
database thread
3 These other tasks will signal when ready for event loop response

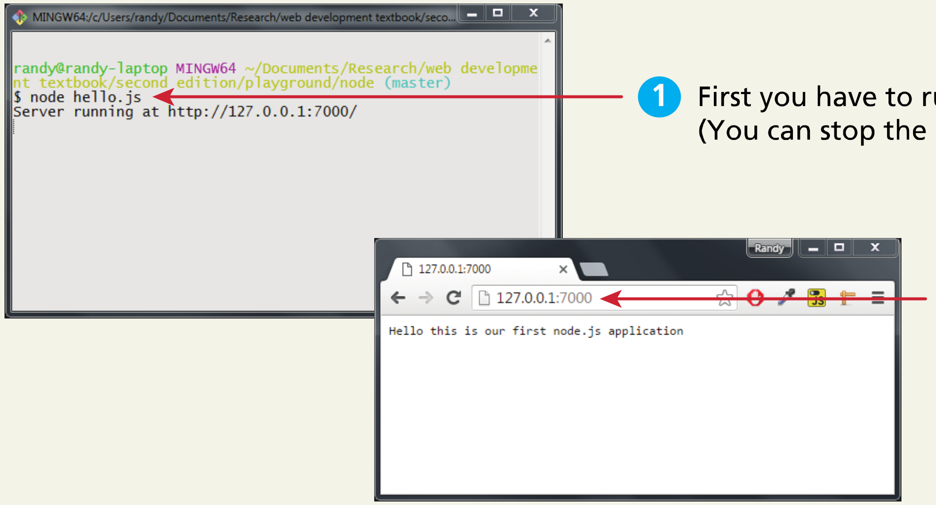
Node.js

Working with Node.js

```
// Load the http module to create an HTTP server
var http = require('http');

// Configure HTTP server to respond with Hello World to all requests
var server = http.createServer(function (request, response) {
        response.writeHead(200, {"Content-Type": "text/plain"});
        response.write("Hello this is our first node.js application\n");
        response.end();
});

// Listen on port 7000 on localhost
server.listen(7000, "localhost");

// display a message on the terminal
console.log("Server running at http://127.0.0.1:7000/");
```

Node.js

Working with Node.js



```
randy@randy-laptop MINGW64 ~/Documents/Research/web developme
nt textbook/second edition/playground/node (master)
$ node hello.js
Server running at http://127.0.0.1:7000/
```

MINGW64:/c/Users/randy/Documents/Research/web development textbook/seco...
127.0.0.1:7000
127.0.0.1:7000
Randy

```
Hello this is our first node.js application
```

1 First you have to run the program via node command (You can stop the program via Ctrl-C)

2 Then use browser to request URL and port

Note: every time you make a change to your Node source file, you will have to stop the program and re-run it.

Node.js

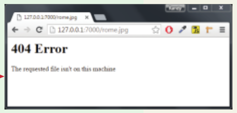
Static file server example

fileserver.js

```
var http = require("http");
var url = require("url");
var path = require("path");
var fs = require("fs");
```

Using two new modules in this example that process URL paths and read/write local files.

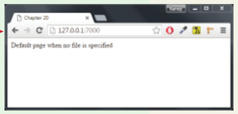
```
// our HTTP server now returns requested files
var server = http.createServer(function (request, response) {

    // get the filename from the URL
    var requestedFile = url.parse(request.url).pathname;
    // now turn that into a file system file name by adding the current
    // local folder path in front of the filename
    var filename = path.join(process.cwd(), requestedFile);

    // check if it exists on the computer
    fs.exists(filename, function(exists) {
        // if it doesn't exist, then return a 404 response
        if (! exists) {
            response.writeHead(404,
               {"Content-Type": "text/html"});
            response.write("<h1>404 Error</h1>\n");
            response.write("The requested file isn't on this machine\n");
            response.end();
            return;
        }

        // if no file was specified, then return default page
        if (fs.statSync(filename).isDirectory())
            filename += '/index.html';

        // file was specified then read it in and send its
        // contents to requestor
        fs.readFile(filename, "binary", function(err, file) {
            // maybe something went wrong ...
            if (err) {
                response.writeHead(500, {"Content-Type": "text/html"});
                response.write("<h1>500 Error</h1>\n");
                response.write(err + "\n");
                response.end();
                return;
            }
            // ... everything is fine so return contents of file
            response.writeHead(200);
            response.write(file, "binary");
            response.end();
        });
    });

});
server.listen(7000, "localhost");
console.log("Server running at http://127.0.0.1:7000/");
```



127.0.0.1:7000/rome.jpg

404 Error

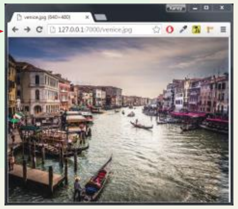
The requested file isn't on this machine



Chapter 20

127.0.0.1:7000

Default page when no file is specified



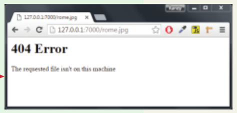
Node.js

Static file server example

fileserver.js

```
var http = require("http");
var url = require("url");
var path = require("path");
var fs = require("fs");

// our HTTP server now returns requested files
var server = http.createServer(function (request, response) {

    // get the filename from the URL
    var requestedFile = url.parse(request.url).pathname;
    // now turn that into a file system file name by adding the current
    // local folder path in front of the filename
    var filename = path.join(process.cwd(), requestedFile);

    // check if it exists on the computer
    fs.exists(filename, function(exists) {
        // if it doesn't exist, then return a 404 response
        if (! exists) {
            response.writeHead(404,
               {"Content-Type": "text/html"});
            response.write("<h1>404 Error</h1>\n");
            response.write("The requested file isn't on this machine\n");
            response.end();
            return;
        }

        // if no file was specified, then return default page
        if (fs.statSync(filename).isDirectory())
            filename += '/index.html';

        // file was specified then read it in and send its
        // contents to requestor
        fs.readFile(filename, "binary", function(err, file) {
            // maybe something went wrong ...
            if (err) {
                response.writeHead(500, {"Content-Type": "text/html"});
                response.write("<h1>500 Error</h1>\n");
                response.write(err + "\n");
                response.end();
                return;
            }
            // ... everything is fine so return contents of file
            response.writeHead(200);
            response.write(file, "binary");
            response.end();
        });
    });

});
server.listen(7000, "localhost");
console.log("Server running at http://127.0.0.1:7000/");
```

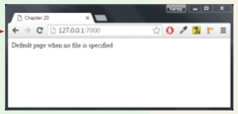
Using two new modules in this example that process URL paths and read/write local files.



127.0.0.1:7000/rome.jpg

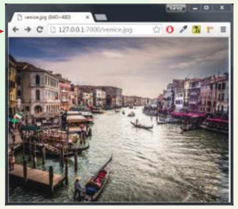
404 Error

The requested file isn't on this machine



127.0.0.1:7000

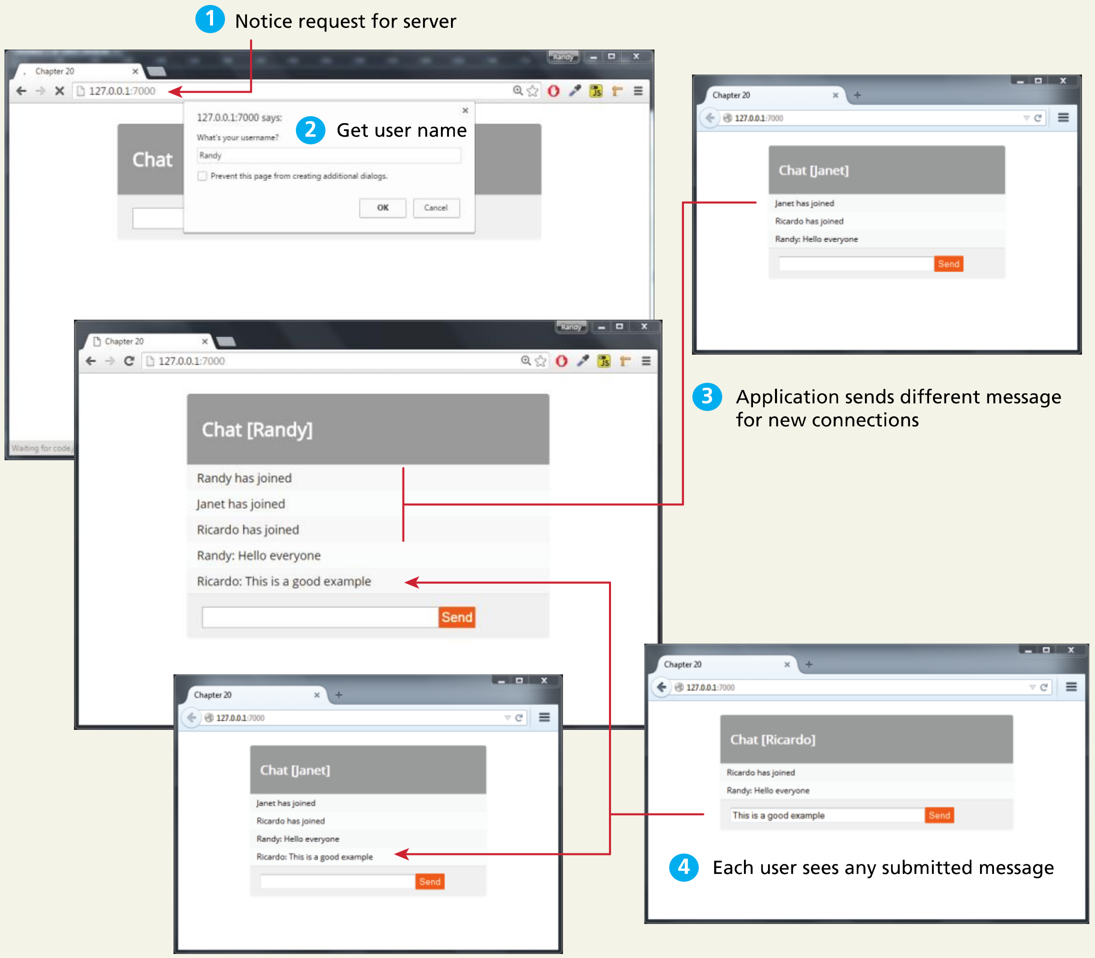
Default page when no file is specified



127.0.0.1:7000/venice.jpg

Node.js

Chat application



Chapter 20

MongoDB

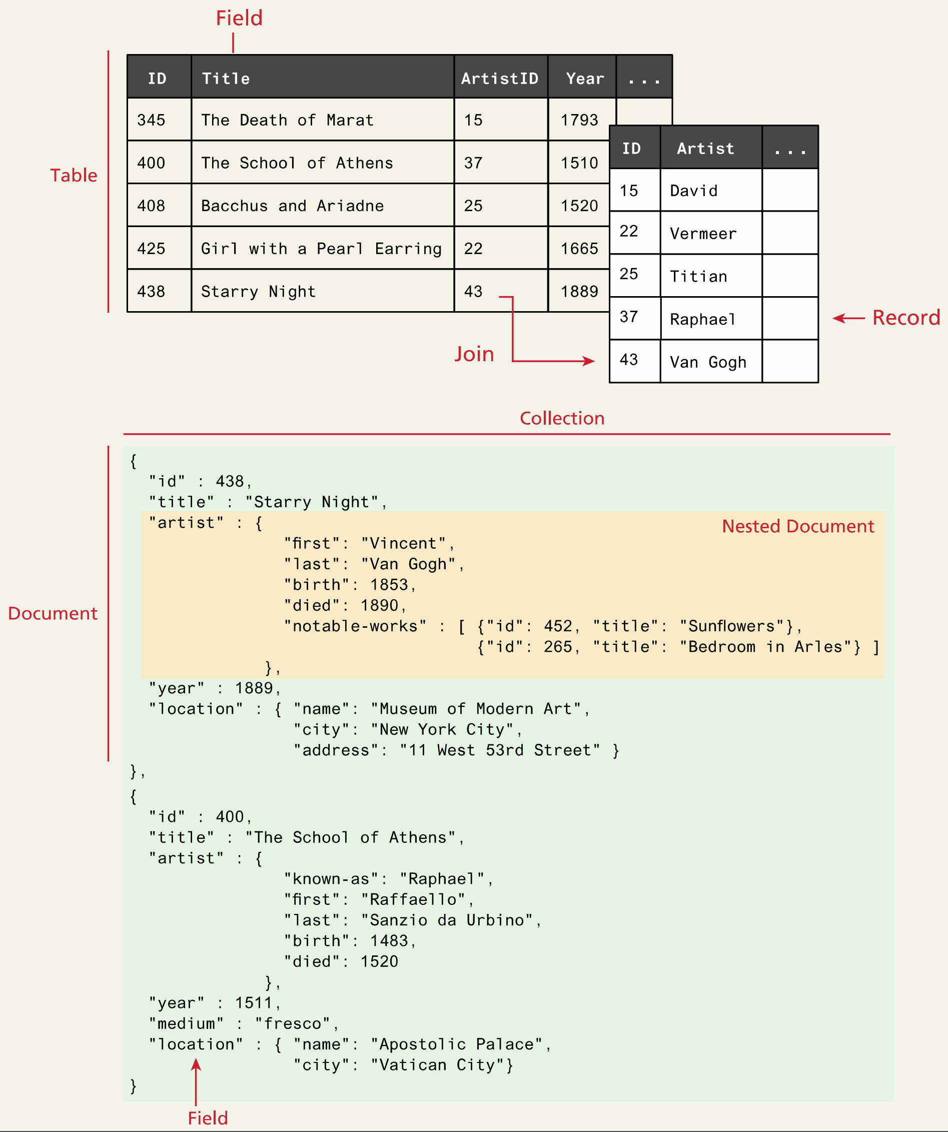
MongoDB Data Model

MongoDB is a document-based database system, and uses different terminology and ideas to describe the way it organizes its data.

- Collections
- Document
- Field
- Nested Document

MongoDB

Comparing to relational DB



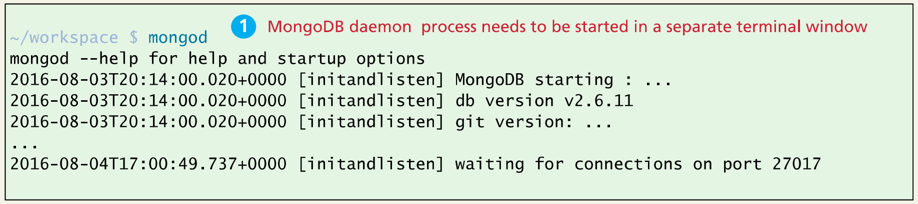
ID	Title	ArtistID	Year	. . .
345	The Death of Marat	15	1793	
400	The School of Athens	37	1510	
408	Bacchus and Ariadne	25	1520	
425	Girl with a Pearl Earring	22	1665	
438	Starry Night	43	1889	

ID	Artist	. . .
15	David	
22	Vermeer	
25	Titian	
37	Raphael	
43	Van Gogh	

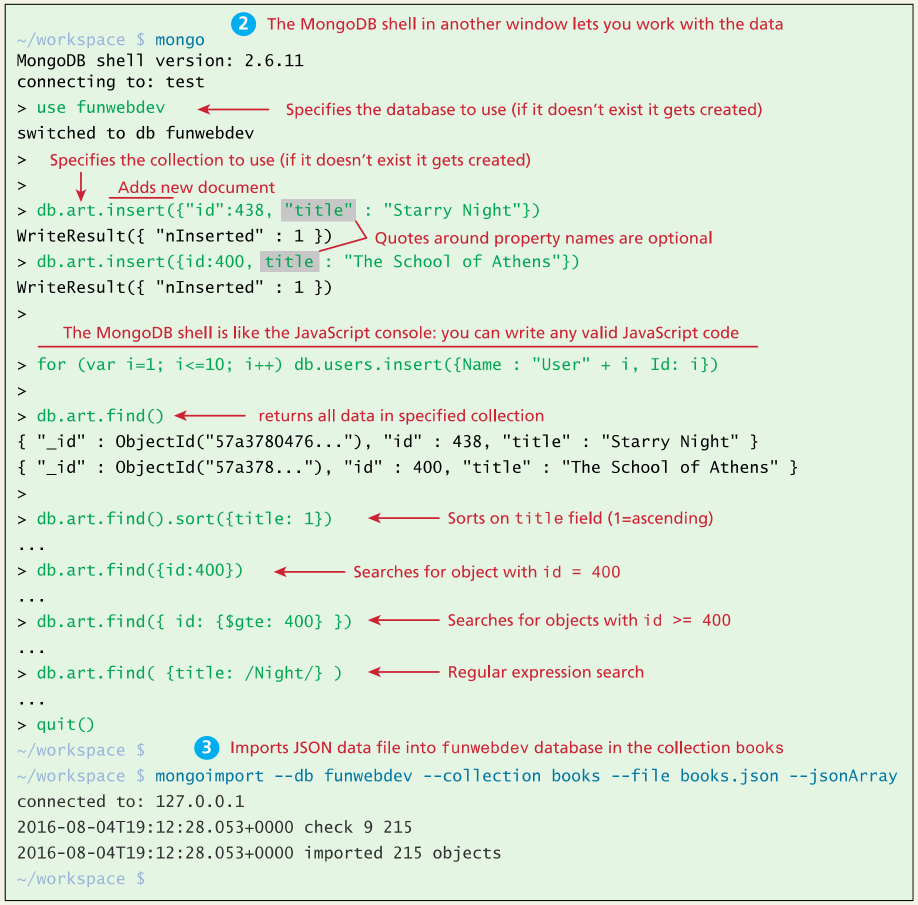
```
{
  "id" : 438,
  "title" : "Starry Night",
  "artist" : {
                "first": "Vincent",
                "last": "Van Gogh",
                "birth": 1853,
                "died": 1890,
                "notable-works" : [ {"id": 452, "title": "Sunflowers"},
                                    {"id": 265, "title": "Bedroom in Arles"} ]
              },
  "year" : 1889,
  "location" : { "name": "Museum of Modern Art",
                 "city": "New York City",
                 "address": "11 West 53rd Street" }
},
{
  "id" : 400,
  "title" : "The School of Athens",
  "artist" : {
                "known-as": "Raphael",
                "first": "Raffaello",
                "last": "Sanzio da Urbino",
                "birth": 1483,
                "died": 1520
              },
  "year" : 1511,
  "medium" : "fresco",
  "location" : { "name": "Apostolic Palace",
                 "city": "Vatican City"}
}
```

MongoDB

Running the MongoDB Shell



```
~/workspace $ mongod        1  MongoDB daemon  process needs to be started in a separate terminal window
mongod --help for help and startup options
2016-08-03T20:14:00.020+0000 [initandlisten] MongoDB starting : ...
2016-08-03T20:14:00.020+0000 [initandlisten] db version v2.6.11
2016-08-03T20:14:00.020+0000 [initandlisten] git version: ...
...
2016-08-04T17:00:49.737+0000 [initandlisten] waiting for connections on port 27017
```



```
                            2  The MongoDB shell in another window lets you work with the data
~/workspace $ mongo
MongoDB shell version: 2.6.11
connecting to: test
> use funwebdev        <------  Specifies the database to use (if it doesn't exist it gets created)
switched to db funwebdev
>  Specifies the collection to use (if it doesn't exist it gets created)
>                   Adds new document
> db.art.insert({"id":438, "title" : "Starry Night"})
WriteResult({ "nInserted" : 1 })    Quotes around property names are optional
> db.art.insert({id:400, title : "The School of Athens"})
WriteResult({ "nInserted" : 1 })
>
   The MongoDB shell is like the JavaScript console: you can write any valid JavaScript code
> for (var i=1; i<=10; i++) db.users.insert({Name : "User" + i, Id: i})
>
> db.art.find()  <------  returns all data in specified collection
{ "_id" : ObjectId("57a3780476..."), "id" : 438, "title" : "Starry Night" }
{ "_id" : ObjectId("57a378..."), "id" : 400, "title" : "The School of Athens" }
>
> db.art.find().sort({title: 1})      <------  Sorts on title field (1=ascending)
...
> db.art.find({id:400})      <------  Searches for object with id = 400
...
> db.art.find({ id: {$gte: 400} })    <------  Searches for objects with id >= 400
...
> db.art.find( {title: /Night/} )     <------  Regular expression search
...
> quit()
~/workspace $        3  Imports JSON data file into funwebdev database in the collection books
~/workspace $ mongoimport --db funwebdev --collection books --file books.json --jsonArray
connected to: 127.0.0.1
2016-08-04T19:12:28.053+0000 check 9 215
2016-08-04T19:12:28.053+0000 imported 215 objects
~/workspace $
```

MongoDB

Comparing a MongoDB query to an SQL query

MongoDB Query

Criteria

Projection

```
db.art.find(
   {
    title: /^The/,
    "artist.died": { $lt: 1800 }
   },
   {
     title: 1,
     year: 1,
     "artist.last": 1,
     "location.name: 1
   }
).sort({year: 1,title : 1}).limit(5)
```

Cursor Modifiers

SQL Equivalent

```
SELECT
   title, year, artist.last,
   location.name
FROM
   art
WHERE
   title LIKE "The%"
   AND
   artist.died < 1800
ORDER BY
   year, title
LIMIT 5
```

MongoDB

Accessing MongoDB Data in Node.js

The official MongoDB driver for Node.js (https://mongodb.github .io/node-mongodb-native/) provides a comprehensive set of methods and properties for accessing a MongoDB database

An ORM (Object-Relational Mapping) tool or framework is a technique for moving data between objects in your programming code and some form of persistence storage (mongoose)

Chapter 20

Angular

Angular is a popular browser-based, open-source JavaScript MVC framework (Goole driven)

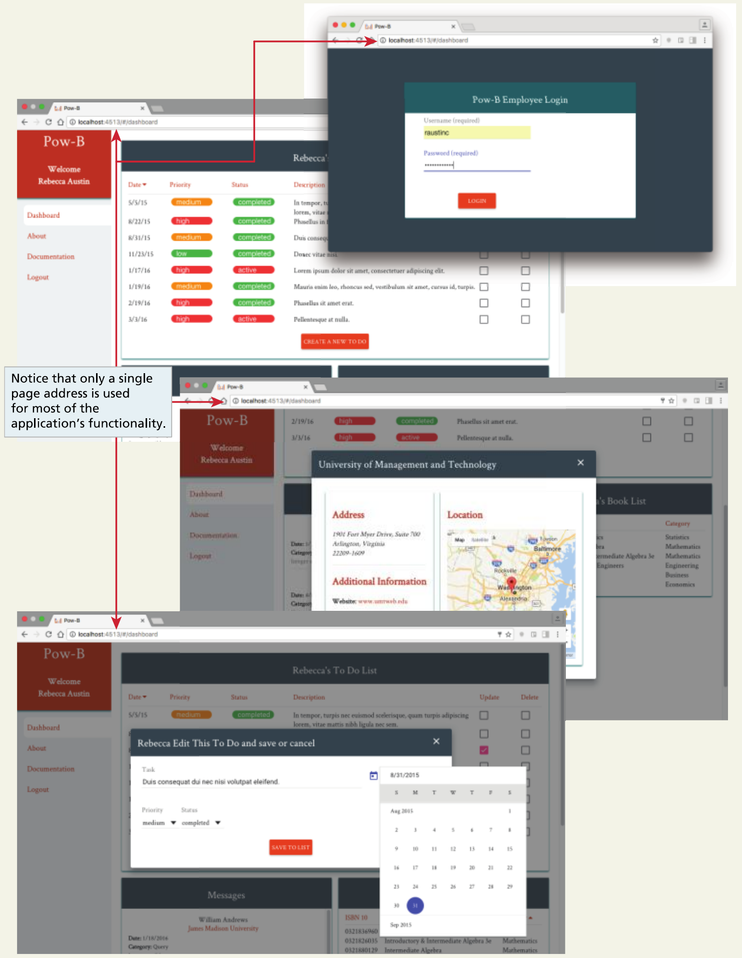
It is the “A” in the MEAN stack, though like everything covered in this chapter, it is independent of the other components of the stack, and can be used without any of them.

Angular 2 now allows developers to write in TypeScript, JavaScript, or Dart.

- many of the online examples and tutorials are TypeScript only
- AngularJS uses JavaScript

Angular

Why AngularJS – Well suited for Single Page Applications


Notice that only a single page address is used for most of the application's functionality.
localhost:4513/#/dashboard
Pow-B Employee Login
Pow-B
Welcome Rebecca Austin
Dashboard
About
Documentation
Logout
University of Management and Technology
Address
Location
Additional Information
Rebecca's To Do List
Rebecca Edit This To Do and save or cancel
SAVE TO LIST
CREATE A NEW TO DO

Angular

Creating a Simple AngularJS Application

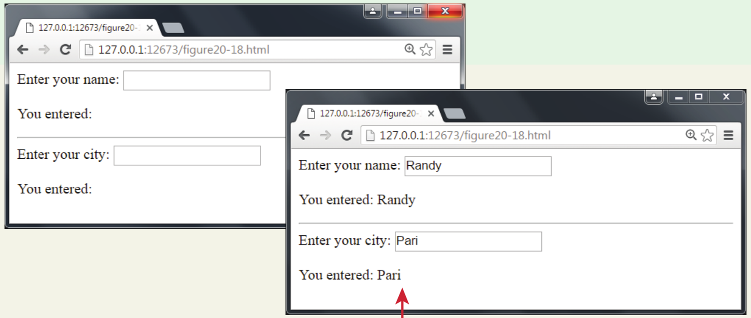
A *directive* for designating the root AngularJS element

A *template*

```
<html ng-app>
<head>
<title>Chapter 20</title>
<script src="https://code.angularjs.org/1.5.0/angular.min.js" >
</script>
</head>
<body>
  Enter your name: <input type="text" ng-model="name" />
  <p>You entered: {{ name }}  </p>

  <hr>
  Enter your city: <input type="text" ng-model="city" />
  <p>You entered: {{ city }}  </p>
</body>
</html>
```

A *directive* for saving the field value in the Model

A *data binding expression*



Enter your name:

You entered:

Enter your city:

You entered:

Enter your name: Randy

You entered: Randy

Enter your city: Pari

You entered: Pari

Appears as user types into textbox

Angular

A Controller

Now this directive is specifying the *module* used in the application

```
<html ng-app="demo">
...
```

This element is going to use a *controller* to get its data

```
<body ng-controller="myController">
    <div id="search">
      City Search: <input type="text" ng-model="search" />
    </div>
```

Save the user's input in a model property named search

```
    <table>
       <tr ng-repeat="city in cities | filter:search | orderBy: 'name'">
          <td>{{city.name }}</td>
           <td>{{city.country}}</td>
       </tr>
    </table>
</body>
</html>
```

A directive to loop through a collection named cities (which is defined in the controller)

Data bind to values in the collection

Uses *filters* to alter how this element works. In this example, the filter filter and the orderBy filter are used to modify how the ng-repeat works. Here the search refers to data item in the model.

A module is an AngularJS container for the different components used in the application

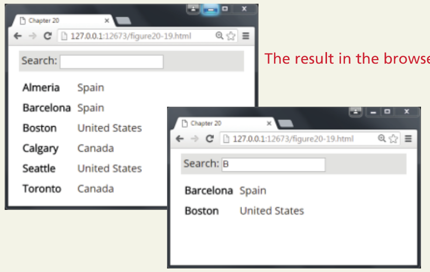
```
var myapp = angular.module('demo',[]);
```

Add a controller to the module named myController

The $scope variable is passed (injected into) the controller by AngularJS

```
myapp.controller('myController', function ($scope) {

     $scope.cities =  [{name: 'Calgary', country: 'Canada'},
                       {name: 'Toronto', country: 'Canada'},
                       {name: 'Boston', country: 'United States'},
                       {name: 'Seattle', country: 'United States'},
                       {name: 'Almeria', country: 'Spain'},
                       {name: 'Barcelona', country: 'Spain'}];
});
```

The $scope variable is used to store the model (data). Here we are defining an array of object literals named cities



Search:

Almeria	Spain
Barcelona	Spain
Boston	United States
Calgary	Canada
Seattle	United States
Toronto	Canada

Search: B

Barcelona	Spain
Boston	United States

The result in the browser (notice the sort order)

The filter filter alters the displayed cities based on the current value of the Search text field.

Angular

A Controller

Now this directive is specifying the *module* used in the application

This element is going to use a *controller* to get its data

Save the user's input in a model property named search

A directive to loop through a collection named cities (which is defined in the controller)

```
<html ng-app="demo">
...
<body ng-controller="myController">
    <div id="search">
      City Search: <input type="text" ng-model="search" />
    </div>
    <table>
       <tr ng-repeat="city in cities | filter:search | orderBy: 'name'">
          <td>{{city.name }}</td>
           <td>{{city.country}}</td>
       </tr>
    </table>
</body>
</html>
```

Uses *filters* to alter how this element works. In this example, the filter filter and the orderBy filter are used to modify how the ng-repeat works. Here the search refers to data item in the model.

Data bind to values in the collection

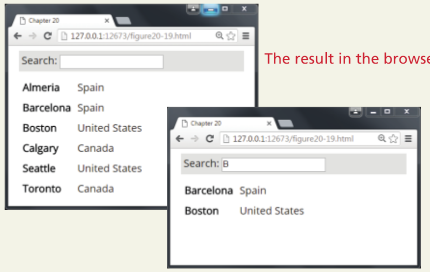
A module is an AngularJS container for the different components used in the application

Add a controller to the module named myController

The $scope variable is passed (injected into) the controller by AngularJS

```
var myapp = angular.module('demo',[]);

myapp.controller('myController', function ($scope) {

     $scope.cities =  [{name: 'Calgary', country: 'Canada'},
                       {name: 'Toronto', country: 'Canada'},
                       {name: 'Boston', country: 'United States'},
                       {name: 'Seattle', country: 'United States'},
                       {name: 'Almeria', country: 'Spain'},
                       {name: 'Barcelona', country: 'Spain'}];
});
```

The $scope variable is used to store the model (data). Here we are defining an array of object literals named cities



```
Search:
Almeria    Spain
Barcelona  Spain
Boston     United States
Calgary    Canada
Seattle    United States
Toronto    Canada
```

```
Search: B
Barcelona  Spain
Boston     United States
```

The result in the browser (notice the sort order)

The filter filter alters the displayed cities based on the current value of the Search text field.

Chapter 20

Summary

Key Terms

- Angular
- build tools
- clickstream
- commodity servers
- context switching
- DIRT (data-intensive real-time) applications
- Ember
- failover clustering
- full-duplex
- MEAN stack
- module
- multiple master replication
- node.js
- npm (Node Package Manager)
- ORM (Object-Relational Mapping)
- push-based web applications
- React
- routing
- sharding
- single master replication
- Single-Page Applications (SPA)
- software framework
- task runner tools
- WebSockets

Summary

Questions?